Árvore Binária de Busca

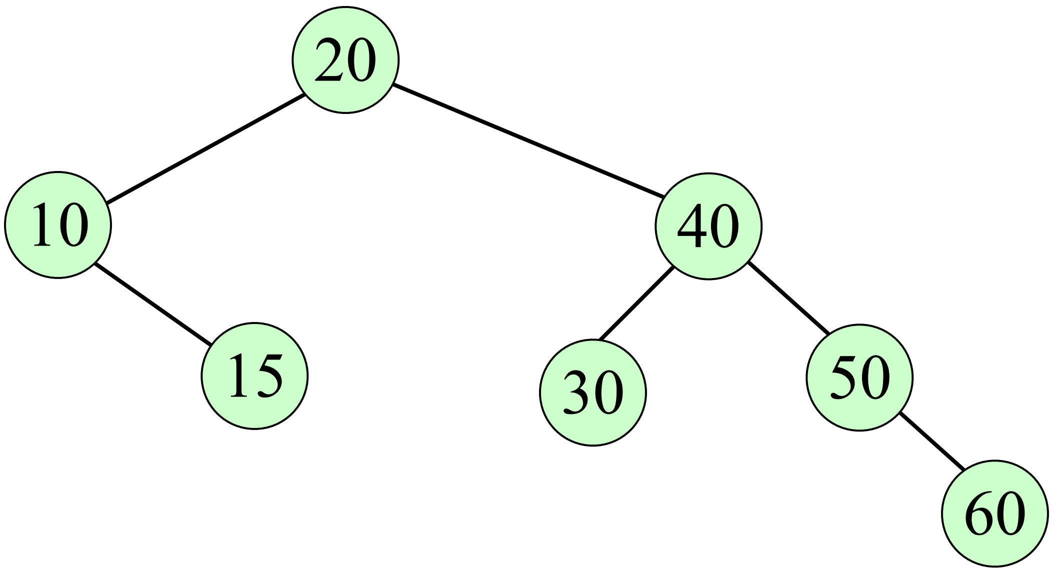
Árvore Binária de Busca

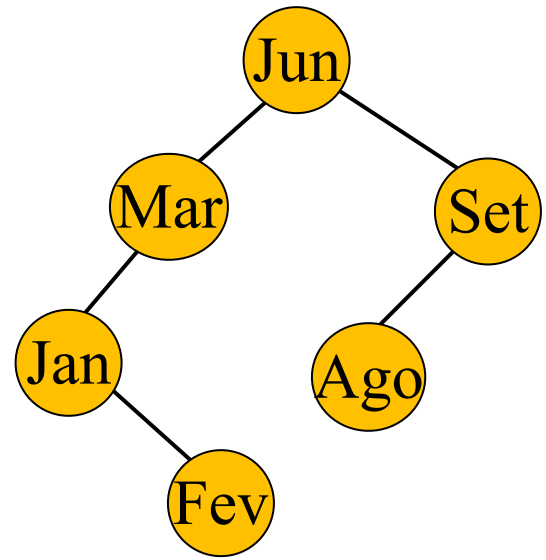
Uma árvore binária de busca é uma árvore binária tal que:

i) A raiz possui uma chave
ii) As chaves dos nós da subárvore esquerda da raiz são menores que a chave da raiz.
iii) As chaves dos nós da subárvore direita da raiz são maiores que a chave da raiz.
iv) As subárvores esquerda e direita são árvores binárias de busca

Árvore Binária de Busca

Exemplos





Operações básicas:

- **Busca**
- **Inserção**
- **Remoção**

Árvore Binária de Busca

Busca

Considere $S = \{s_1,...,s_n\}$ um conjunto de n chaves tais que

$s_1 < ... < s_n$.

Dado um valor x, verificar se $x \in S$.

Caso $x = s_i$, $s_i \in S$, retornar i.

Árvore Binária de Busca

Qual a complexidade da busca sequencial em um lista linear?

Qual a complexidade da busca binária em um lista linear?

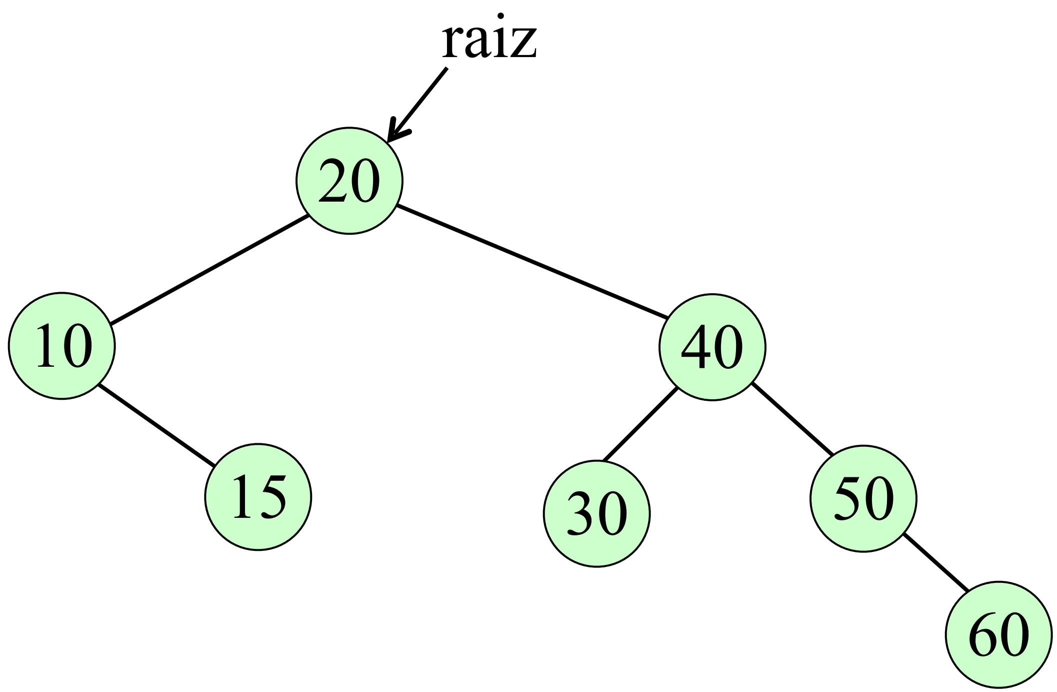
Qual a complexidade da busca em uma árvore binária?

Qual a complexidade da busca em uma árvore binária de busca?

Árvore Binária de Busca

Exemplo Busca

Chaves de busca: 15, 35



Árvore Binária de Busca

Algoritmo Busca

Busca_arvbb(raiz, *x*, 0)

```
Busca_arvbb(pont_no pt, int x, int f)
 se (pt ≠ λ)
     se (pt↑.chave = x) f ← 1
     senão
         se (x < pt↑.chave)
              se (pt↑.esq = λ) f ← 2
              senão
                   pt ← pt↑.esq
                   Busca_arvbb(pt, x, f)
         senão
              se (pt↑.dir = λ) f ← 3
              senão
                   pt ← pt↑.dir
                   Busca_arvbb(pt, x, f)
```

f **=** 0, árvore vazia

f **=** 1, chave encontrada e *pt* aponta para nó onde a chave se encontra

f **=** 2, chave não encontrada e *pt* aponta para nó cuja árvore esquerda é vazia

f **=** 3, chave não encontrada e *pt* aponta para nó cuja árvore direita é vazia

Árvore Binária de Busca

Complexidade da Busca

Melhor Caso: Chave é encontrada na raiz

Árvore Binária de Busca

Complexidade da Busca

Pior Caso: Chave não encontrada (depois de percorrer o maior caminho entre a raiz e uma folha)

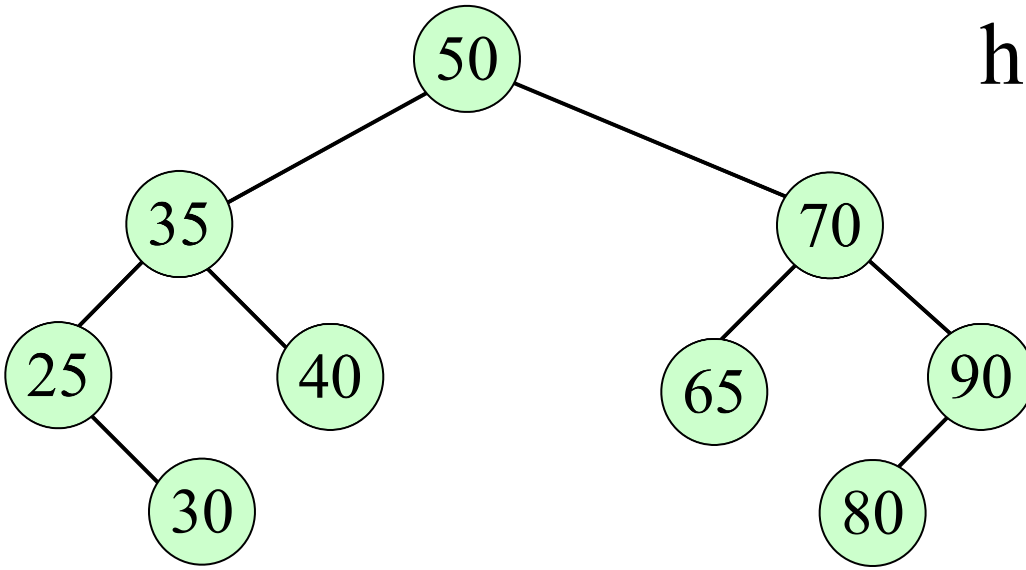
=

$h(T)$

Quais são os casos para $h(T)$?

Árvore Binária de Busca

Árvore Binária Completa



$$h(T) = 1 + \lfloor \log n \rfloor.$$

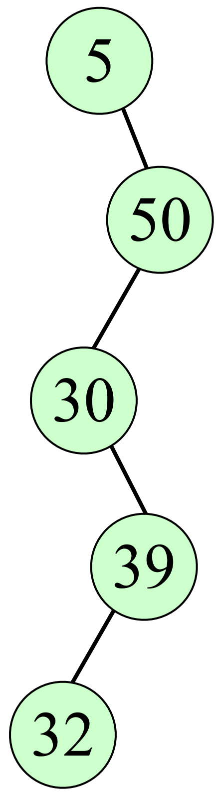
$$h(T) = O(\log n)$$

Lema 2

Considere T uma árvore binária completa com $n > 0$ nós.
Então T possui altura h mínima.
Além disso, $h = 1 + \lfloor \log n \rfloor$.

Árvore Binária de Busca

Árvore Zig-zag



$$h(T) = n$$
$$h(T) = O(n)$$

Árvore Binária de Busca

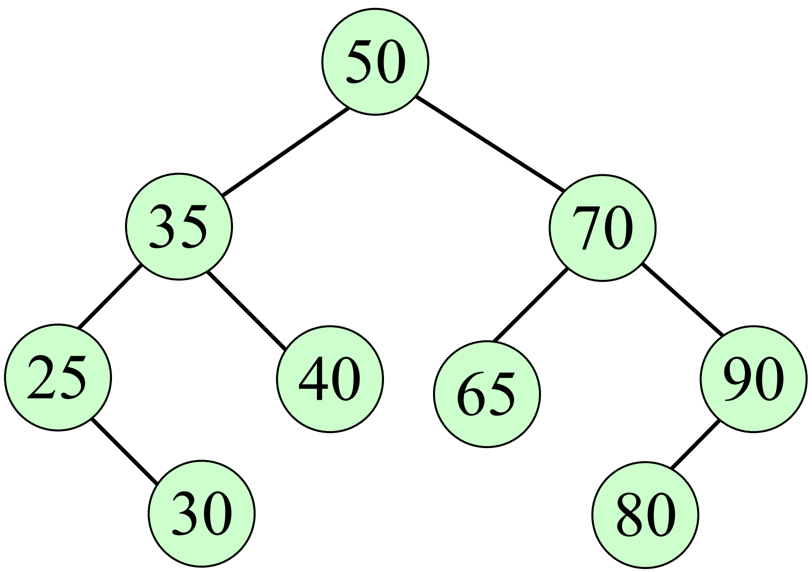
Complexidade da Busca

A complexidade da busca depende da altura da árvore. Sendo assim, é interessante construir a árvore com a menor altura possível. Como visto anteriormente, a árvore que possui altura mínima para um conjunto de n chaves é a ***árvore completa***. Nesse caso, a complexidade do algoritmo é $O(\log n)$.

Árvore Binária de Busca

Busca por sucessor

Dado um nó **v**, buscar o seu sucessor



Qual o sucessor de 50?
Qual o sucessor de 70?
Qual o sucessor de 40?

Árvore Binária de Busca

Busca por sucessor

Dado um nó **v**, buscar o seu sucessor

sucessor (v)

se v↑.dir ≠ λ então
   retorne min(v↑.dir)
senão
   pai ← v↑.pai
   enquanto pai ≠ λ e v==pai↑.dir faça
      v ← pai
      pai ← v↑.pai
   retorne pai

Um nó **p** é sucessor de **v** se:

p é o menor entre os descendentes direitos de **v**

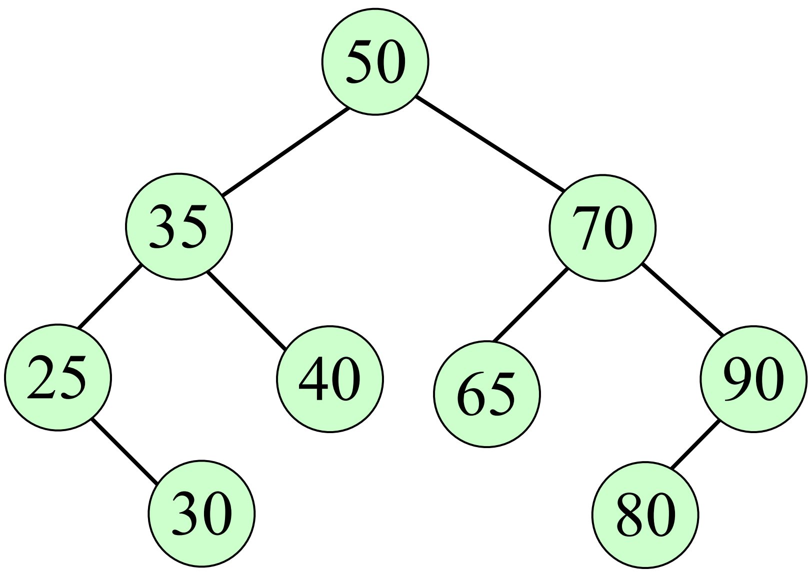
p é o ancestral « mais baixo » de **v** tal que **v** é descendente **esquerdo** de **p**

E se **v** não possuir descentes direitos e não possuir ancestrais?

Árvore Binária de Busca

Busca por antecessor

Dado um nó **v**, buscar o seu antecessor



Qual o antecessor de 50?
Qual o antecessor de 70?
Qual o antecessor de 65?

Árvore Binária de Busca

Busca por antecessor

Dado um nó **v**, buscar o seu antecessor

```
antecessor (v)
se v↑.esq ≠ λ então
    retorne max(v↑.esq)
senão
    pai ← v↑.pai
    enquanto pai ≠ λ  e  v==pai↑.esq faça
        v ← pai
        pai ← v↑.pai
   retorne pai
```

Um nó **p** é antecessor de **v** se:

p é o maior entre os descendentes esquerdos de **v**

p é o ancestral « mais baixo » de **v** tal que **v** é descendente **direito** de **p**

E se **v** não possuir descentes esquerdos e não possuir ancestrais?

Árvore Binária de Busca

Inserção

O novo nó é inserido no lugar de uma subárvore vazia.

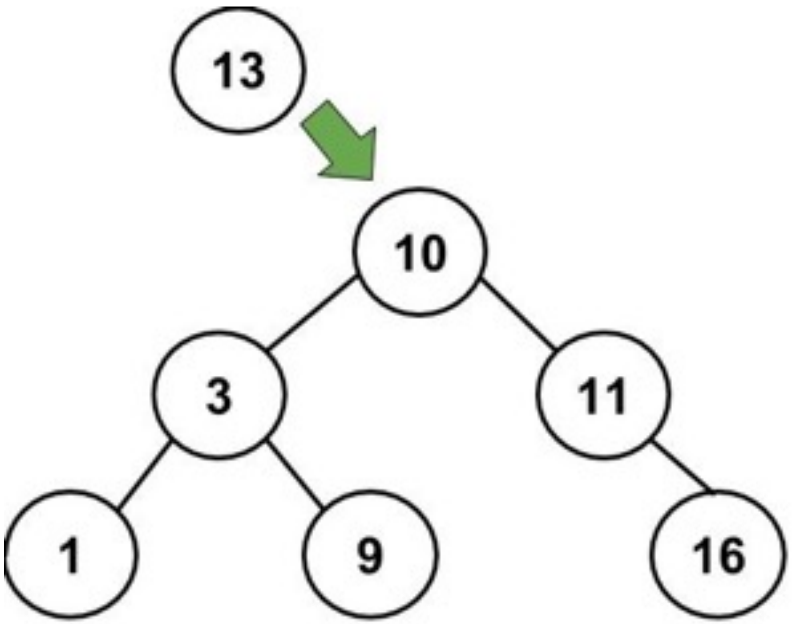
```
inserção(x, pt, f)
    pt ← ptraiz
    busca_árvore(x, pt, f)
    se (f = 1) então "Elemento já existe"
    senão
        ocupar(pt1); pt1↑.chave ← x; pt1↑.esq = λ; pt1↑.dir= λ
        se (f = 0) então ptraiz ← pt1
        senão
            se (f = 2) então pt↑.esq = pt1
            senão pt↑.dir = pt1
```

Árvore Binária de Busca

Inserção

Complexidade?

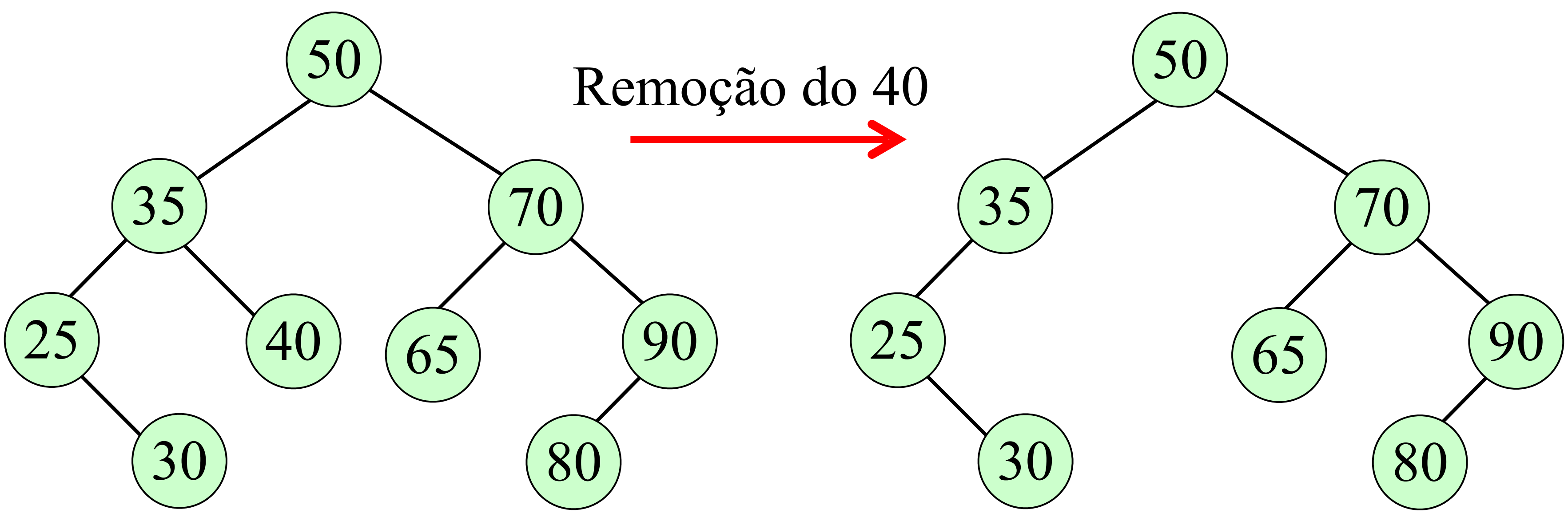
Dominada pela complexidade da busca



Árvore Binária de Busca

Remoção

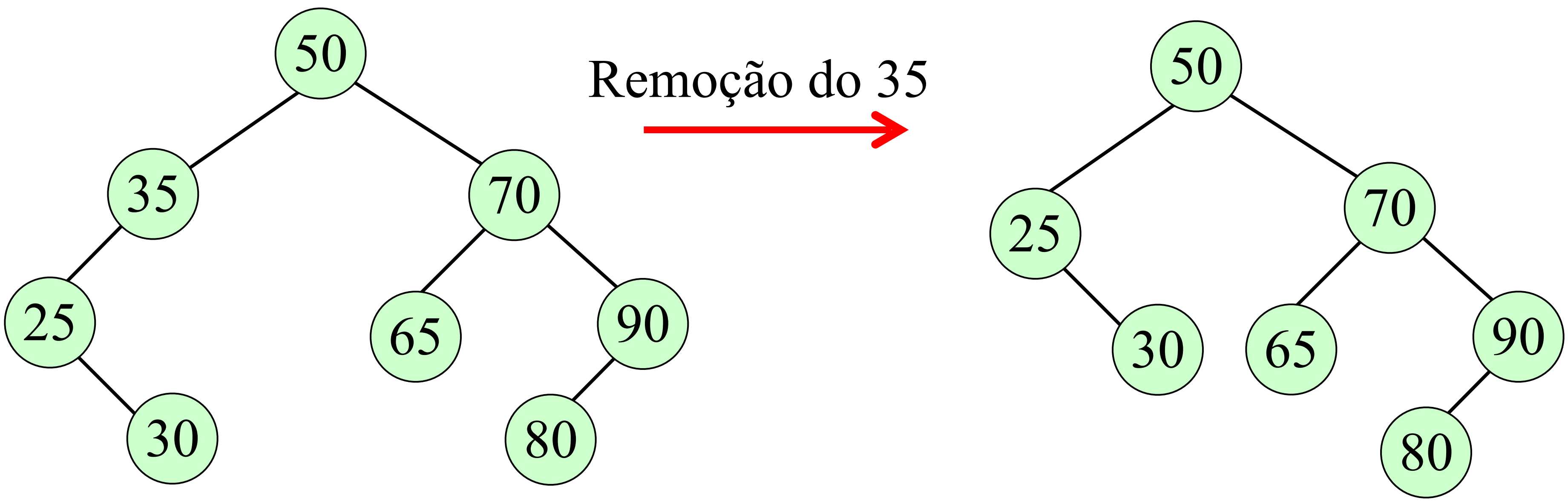
Caso 1. Deseja-se remover uma chave que está numa folha.



Árvore Binária de Busca

Remoção

Caso 2. A chave removida não é uma folha, mas possui **uma** subárvore vazia.



Árvore Binária de Busca

Remoção

Caso 3. A chave removida não é uma folha e possui duas subárvores não vazias.

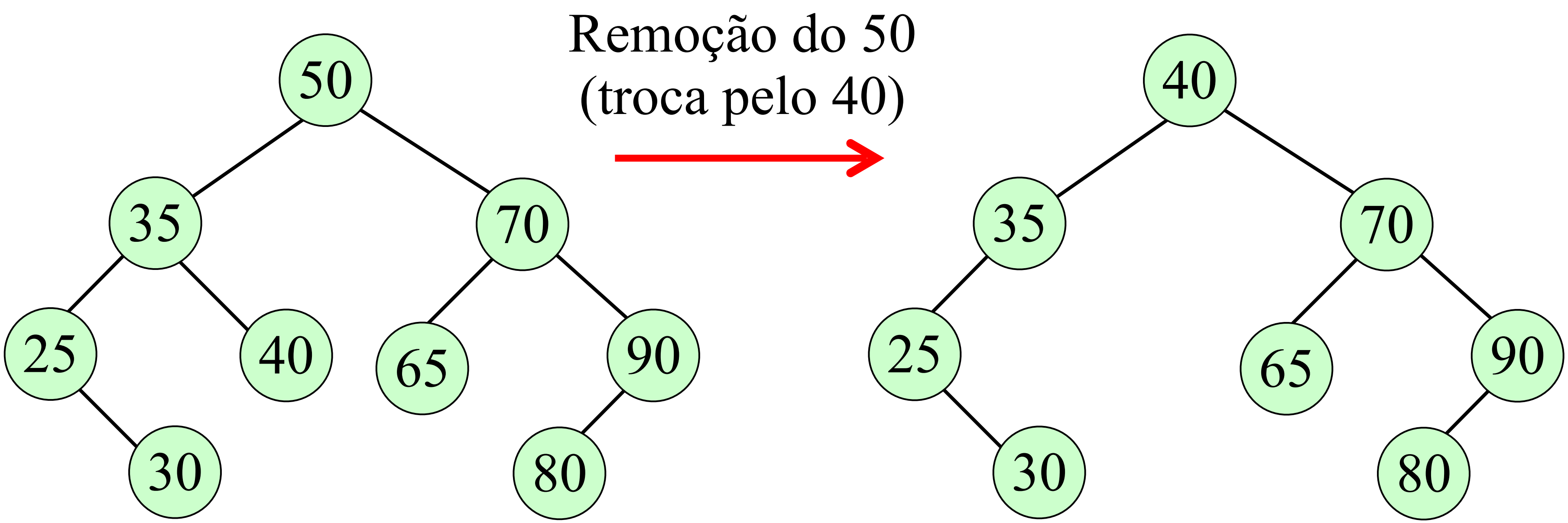
Neste caso, transformar na remoção dos casos 1 ou 2.

Isto é feito substituindo o elemento a ser removido pelo **maior elemento da subárvore esquerda (<u>antecessor</u>)** ou o **menor da subárvore direita (<u>sucessor</u>)**.

Árvore Binária de Busca

Remoção

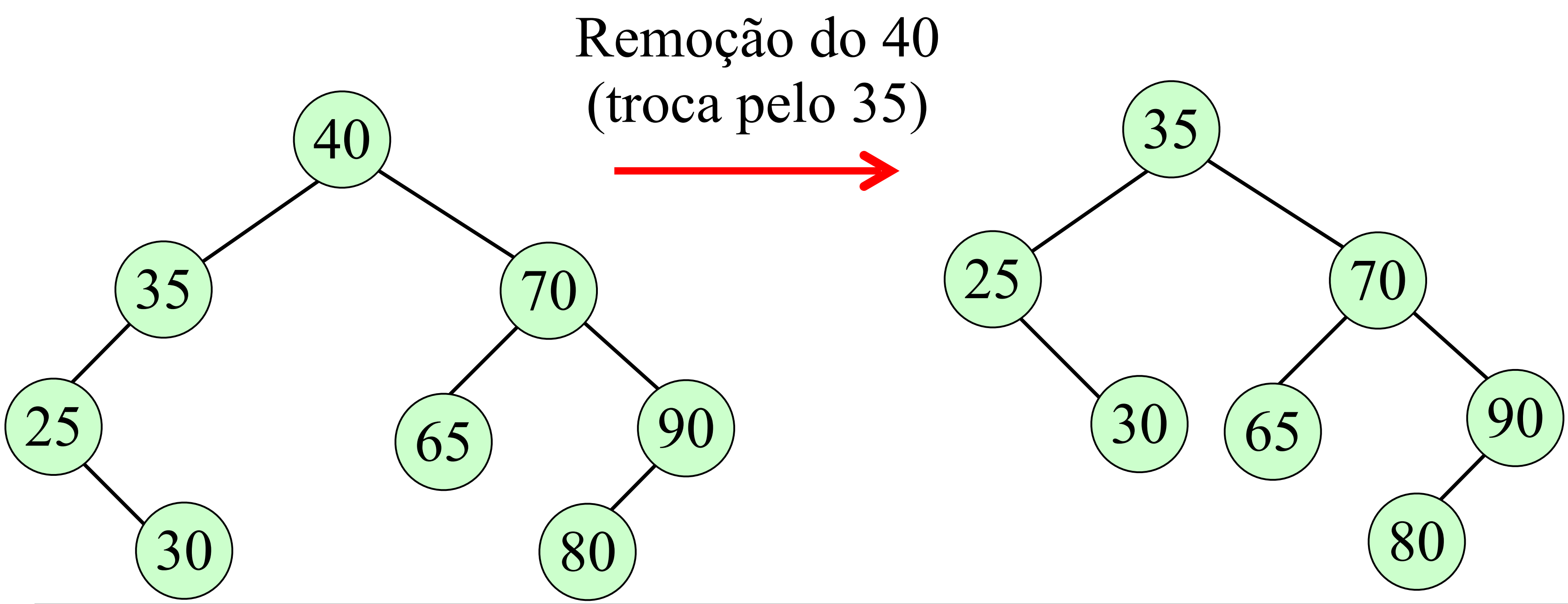
Caso 3. A chave removida não é uma folha e possui duas subárvores não vazias.



Árvore Binária de Busca

Remoção

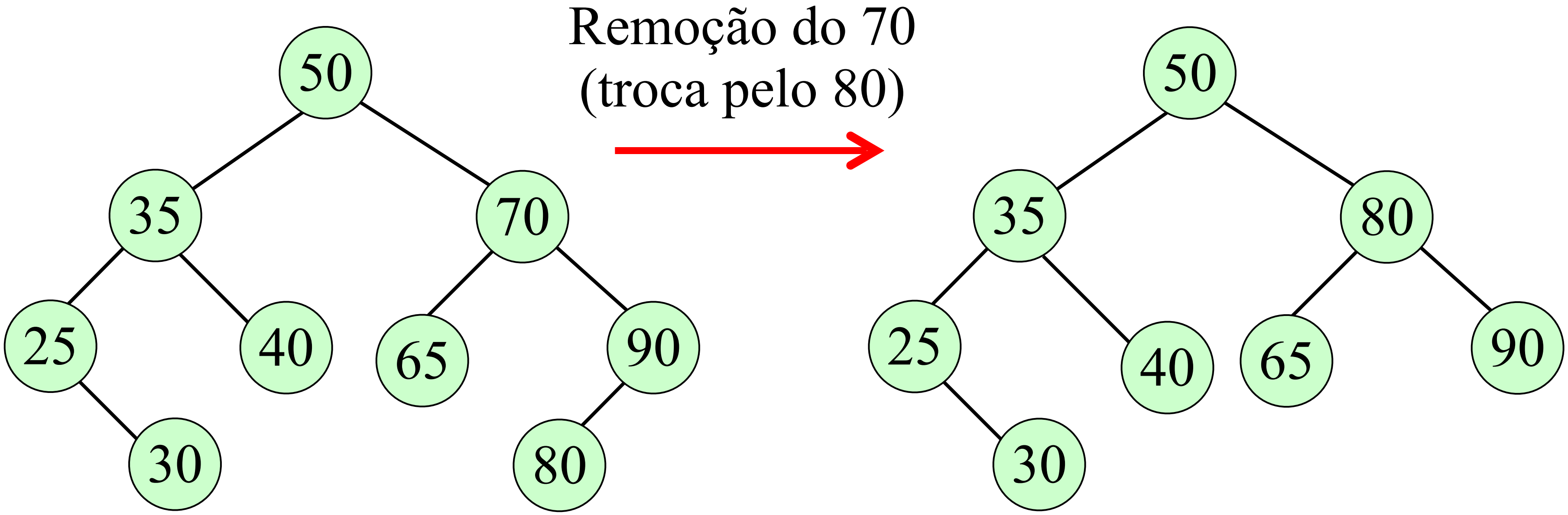
Caso 3. A chave removida não é uma folha e possui duas subárvores não vazias.



Árvore Binária de Busca

Remoção

Caso 3. A chave removida não é uma folha e possui duas subárvores não vazias.



Árvore Binária de Busca

Remoção

Complexidade?

Dominada pela complexidade da busca

Exercício

Escrever o algoritmo de remoção em árvore binária de busca.

Árvore Binária de Busca Ótima

Árvore Binária de Busca Ótima

Comprimento de Caminho Interno

Para buscar uma chave, s_k, o algoritmo percorre um caminho da raiz até o nó s_k.

Dado um conjunto de n chaves $\{s_1,\ldots,s_n\}$ devemos considerar quantas comparações no total o algoritmo efetuará para localizar todas as chaves.

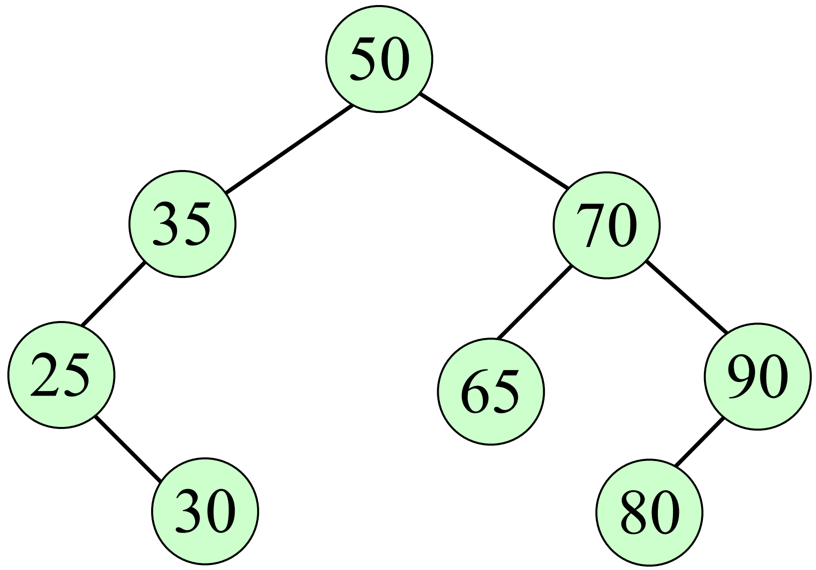
Para cada chave s_k o algoritmo fará nível(s_k) comparações.

O algoritmo fará $\sum_{k=1}^{n} nível(s_k)$ comparações.

Este valor é denominado *comprimento do caminho interno* de T, I(T).

Árvore Binária de Busca Ótima

Comprimento de Caminho Interno

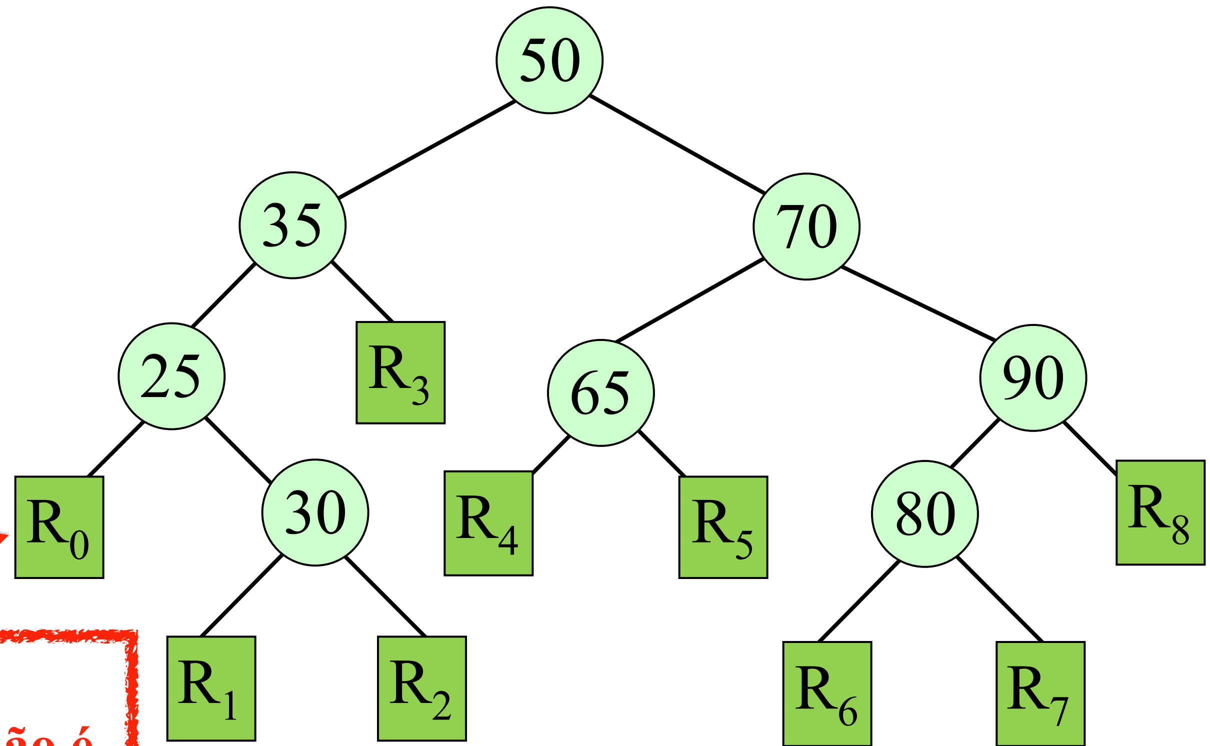


$$I(T) = 1(1) + 2(2) + 3(3) + 2(4) = 22$$

Número de comparações para localizar todas as chaves

Árvore Binária de Busca Ótima

Árvore com nós externos



Um nó externo é, na verdade, fictício. Ele não é implementado de fato, mas apenas simboliza um caso onde a busca pode falhar.

Árvore Binária de Busca Ótima

Comprimento de Caminho Externo

Quando se efetua uma busca **sem sucesso** de uma chave s_k o algoritmo percorre um caminho da raiz até um nó externo R_k e o número de comparações é nível(R_k) – 1.

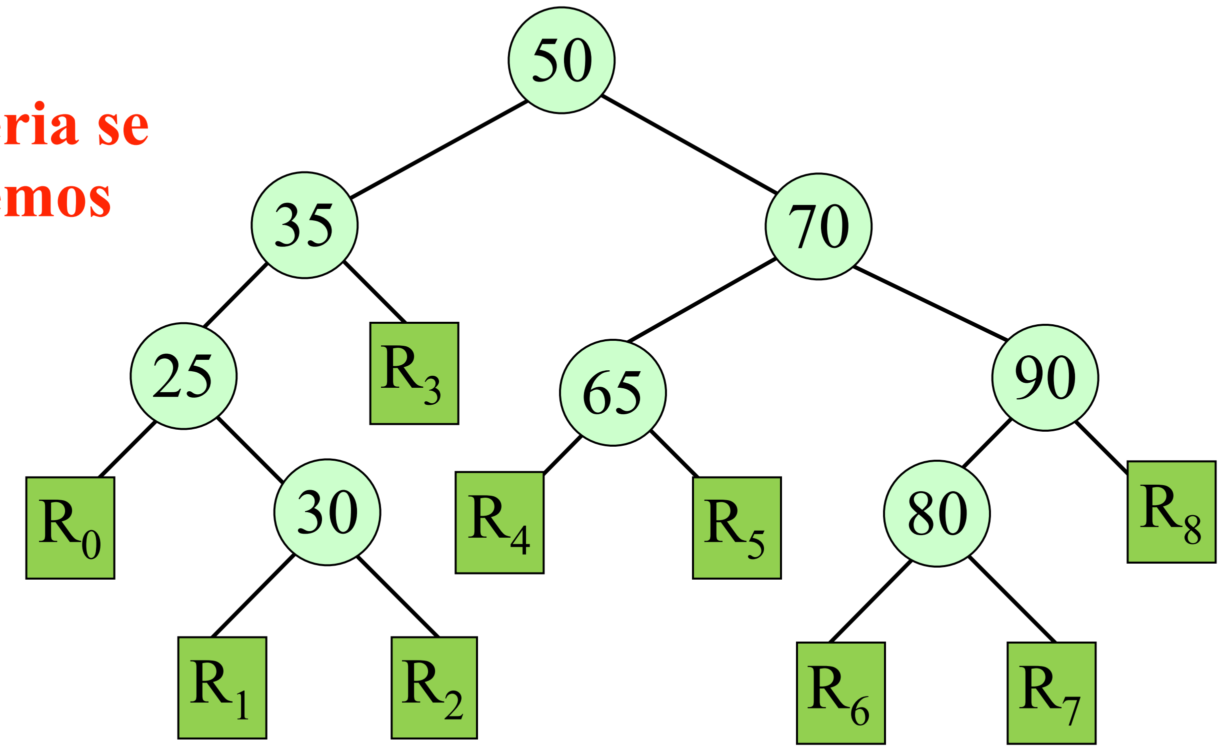
Assim, analogamente ao comprimento de caminho interno, define-se comprimento de caminho externo como

$$E(T) = \sum_{k=0}^{n} nível(R_k) - 1$$

Árvore Binária de Busca Ótima

Comprimento de Caminho Externo

O que aconteceria se buscássemos o 40?



$$E(T) = 1(2) + 4(3) + 4(4) = 30$$

Exercício

Prove que

$$E(T) = I(T) + n$$

… onde n é a quantidade de nós de T.

Árvore Binária de Busca Ótima

Frequências de Acesso Diferenciadas

Considere que cada chave s_i, $i = 1, \ldots, n$, possui probabilidade de acesso p_i, tal que $p_1, p_2, \ldots, p_n$, se distribuem de forma qualquer.

Além disso, os nós externos $R_0, \ldots, R_n$, possuem probabilidades p_0', $p_1', \ldots, p_n'$. Dadas as probabilidades, deseja-se construir a ***Árvore Ótima*** para efetuar a busca.

Obs. Se $p_1 = p_2 = \ldots = p_n = p_0' = p_1' = \ldots = p_n'$, então a *árvore ótima* é a *árvore completa.*

Árvore Binária de Busca Ótima

Busca com Sucesso

São necessárias nível(s_k) comparações para encontrar s_k.

A probabilidade de acesso a s_k é p_k.

A contribuição deste nó é p_k nível(s_k).

Para todos os nós internos tem-se:

$$\sum_{i=1}^{n} p_i nível(s_i) \longrightarrow$$

Comprimento de Caminho Interno Ponderado

Árvore Binária de Busca Ótima

Busca sem Sucesso

$$\sum_{i=0}^{n} p_i^{'}\left(nível\left(R_i\right)-1\right) \longrightarrow \text{Comprimento de Caminho Externo Ponderado}$$

Árvore Binária de Busca Ótima

Árvore Ótima

Sua construção **minimiza** a expressão ($c(T)$ é o custo da árvore)

$$c(T) = \sum_{i=1}^{n} p_i \, nível(s_i) + \sum_{i=0}^{n} p'_i \left(nível(R_i) - 1 \right)$$

Uma árvore binária de busca ótima (ou custo ótimo) é a árvore binária de busca que minimiza o custo dado pela expressão acima!!!

Note que toda árvore binária de busca ótima é, necessariamente, uma árvore binária de busca, mas o inverso não é verdadeiro!!!!

Árvore Binária de Busca Ótima

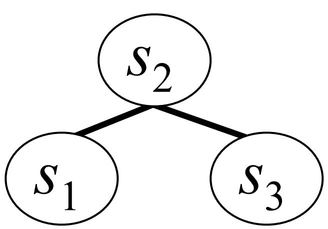
Árvore Ótima

$$c(T) = \sum_{i=1}^{n} p_i\,nível(s_i) + \sum_{i=0}^{n} p_i'\,(nível(R_i) - 1)$$

Exemplo

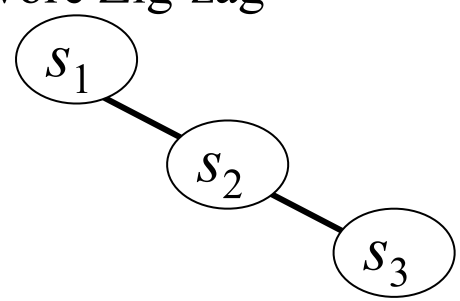
Dados s_1, s_2 e s_3, tais que $s_1 < s_2 < s_3$ e $p_1 = 0{,}8$, $p_2 = 0{,}1$, $p_3 = 0{,}1$, $p_0' = p_1' = p_2' = p_3' = 0$

- Árvore Completa



$c(T) = 0{,}8(2) + 0{,}1(1) + 0{,}1(2) = 1{,}9$

- Árvore Zig-zag



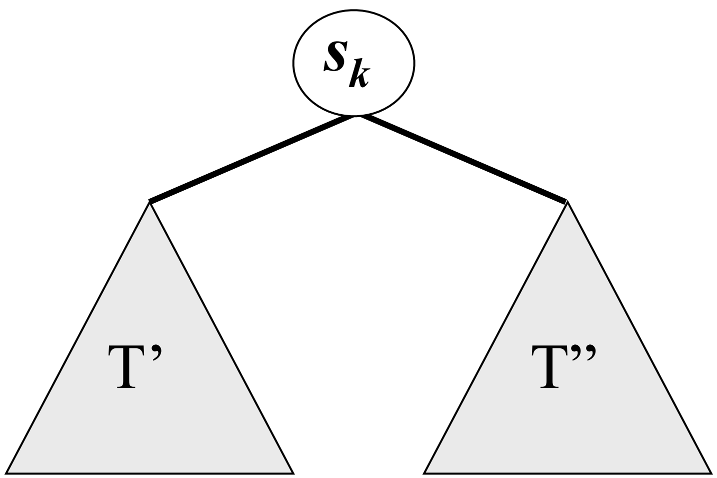
$c(T) = 0{,}8(1) + 0{,}1(2) + 0{,}1(3) = 1{,}3$

Árvore Binária de Busca Ótima

Árvore Ótima

Considere o conjunto de chaves $\{s_1, ..., s_n\}$ s_i, $i = 1, ...,n$.

Suponha que a raiz s_k da árvore ótima é conhecida.



T' $\rightarrow$ subárvore binária de busca $\{s_1, ..., s_{k-1}\}$

T'' $\rightarrow$ subárvore binária de busca $\{s_{k+1}, ..., s_n\}$

Árvore Binária de Busca Ótima

Árvore Ótima

Lema. As subárvores de uma árvore binária de busca ótima, são também ótimas.

Prova. Se isso não ocorresse, então a substituição da subárvore não ótima pela ótima, resultaria em uma diminuição do custo total, o que é uma contradição com o fato da árvore ser ótima. Portanto, T’ e T” são árvores ótimas.

Árvore Binária de Busca Ótima

Árvore de Custo Mínimo

Questões:

1. Como determinar s_k?

2. Como construir T' e T''?

- Para resolver 1, a ideia é tentar todas as $O(n)$ alternativas.

- Para resolver 2, usar a recursão.

Árvore Binária de Busca Ótima

Seja $c(T)$ o custo da árvore T.

É necessário encontrar uma relação entre os custos $c(T)$, $c(T')$ e $c(T'')$.

Sabe-se que:

$$c(T) = \sum_{i=1}^{n} p_i \, nível(s_i) + \sum_{i=0}^{n} p'_i \left(nível(R_i) - 1 \right)$$

Árvore Binária de Busca Ótima

$$c(T) = \sum_{i=1}^{n} p_i \, nível(s_i) + \sum_{i=0}^{n} p'_i \left(nível(R_i) - 1\right)$$

Chamando de:

T(i,j) → a árvore de busca ótima para as chaves $\{s_{i+1}, \ldots, s_j\}$, $i \leq j$.

T(i,i) = 0 e T(0,n) = árvore ótima final

$c(i,j)$ → custo da árvore T(i,j)

P(i,j) → a soma das probabilidades relativas as chaves $\{s_{i+1}, \ldots, s_j\}$

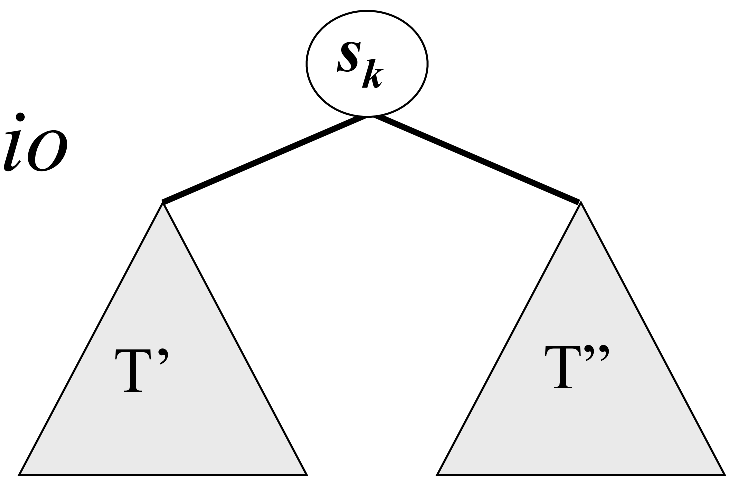
$$P(i, j) = \sum_{k=i+1}^{j} p_k + \sum_{k=i}^{j} p'_k$$

Árvore Binária de Busca Ótima

Lema. Se s_k é a raiz de T, então para $i < j$, tem-se:

$$c(i,j) = c(i, k\text{-}1) + c(k,j) + \mathrm{P}(i,j)$$

Prova: Exercício



$$P(i,j) = \sum_{k=i+1}^{j} p_k + \sum_{k=i}^{j} p'_k$$

Árvore Binária de Busca Ótima

O lema leva à recorrência:

$$\begin{cases} c(i,i) = 0 \\ c(i,j) = \min\limits_{i<k\leq j} \{c(i,k-1) + c(k,j)\} + P(i,j) \end{cases}$$

c(i,j) **e** ***P(i,j)*** **podem ser implementados com matrizes n+1 por n+1**

Número de pares distintos = $n(n+1)/2 \rightarrow O(n^2)$

Árvore Binária de Busca Ótima

Algoritmo

Ao final, $c(0,n)$ dará o custo da árvore T com n nós

Para $j \leftarrow 0$ até n faça
  $c[j,j] \leftarrow 0$; $P[j,j] \leftarrow p'_j$

Para $d \leftarrow 1$ até n faça
  para $i \leftarrow 0$ até n-d faça
    $j \leftarrow i + d$
    $P[i,j] \leftarrow P[i,j\text{-}1] + p_j + p'_j$
    $c[i,j] \leftarrow \min_{i < k \leq j} \{ c[i,k\text{-}1] + c[k,j]\} + P[i,j]$

Complexidade $\Theta(n^3)$

Árvore Binária de Busca Ótima

Exercício - simule a execução do algoritmo anterior para o seguinte caso de teste.

j	0	1	2	3	4
p_j	-	10	1	3	2
p'_j	2	1	1	1	1

$$P = \left| \begin{array}{ccccc} 2 & & & & \\ - & 1 & & & \\ - & - & 1 & & \\ - & - & - & 1 & \\ - & - & - & - & 1 \end{array} \right| \qquad c = \left| \begin{array}{ccccc} 0 & & & & \\ - & 0 & & & \\ - & - & 0 & & \\ - & - & - & 0 & \\ - & - & - & - & 0 \end{array} \right|$$